mathematics

MDPI

Article

Population-Based Redundancy Control in Genetic Algorithms: Enhancing Max-Cut Optimization

Yong-Hyuk Kim [1], Zong Woo Geem [2] and Yourim Yoon [3,*]

[1] School of Software, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 01897, Republic of Korea; yhdfly@kw.ac.kr
[2] Department of Smart City, Gachon University, 1342 Seongnamdaero, Sujeong-gu, Seongnam-si 13120, Republic of Korea; geem@gachon.ac.kr
[3] Department of Computer Engineering, Gachon University, 1342 Seongnamdaero, Sujeong-gu, Seongnam-si 13120, Republic of Korea
* Correspondence: yryoon@gachon.ac.kr

Abstract: The max-cut problem is a well-known topic in combinatorial optimization, with a wide range of practical applications. Given its NP-hard nature, heuristic approaches—such as genetic algorithms, tabu search, and harmony search—have been extensively employed. Recent research has demonstrated that harmony search can outperform genetic algorithms by effectively avoiding redundant searches, a strategy similar to tabu search. In this study, we propose a modified genetic algorithm that integrates tabu search to enhance solution quality. By preventing repeated exploration of previously visited solutions, the proposed method significantly improves the efficiency of traditional genetic algorithms and achieves performance levels comparable to harmony search. The experimental results confirm that the proposed algorithm outperforms standard genetic algorithms on the max-cut problem. This work demonstrates the effectiveness of combining tabu search with genetic algorithms and offers valuable insights into the enhancement of heuristic optimization techniques. The novelty of our approach lies in integrating solution-level tabu constraints directly into the genetic algorithm's population dynamics, enabling redundancy prevention without additional memory overhead, a strategy not previously explored in the proposed hybrids.

Keywords: genetic algorithm; tabu search; max-cut problem; hybrid metaheuristics; redundant search prevention; combinatorial optimization

MSC: 68T20; 68W50; 90C27

Academic Editor: Ioannis Tsoulos

Received: 28 March 2025
Revised: 19 April 2025
Accepted: 24 April 2025
Published: 25 April 2025

Citation: Kim, Y.-H.; Geem, Z.W.; Yoon, Y. Population-Based Redundancy Control in Genetic Algorithms: Enhancing Max-Cut Optimization. *Mathematics* **2025**, *13*, 1409. https://doi.org/10.3390/math13091409

1. Introduction

The max-cut problem is a fundamental problem in combinatorial optimization. Formally, given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the objective is to partition the vertex set V into two disjoint subsets S and $V \setminus S$ such that the number (or total weight) of edges crossing between the two subsets is maximized. This leads to the definition of the cut value:

$$\mathrm{cut}(S) = \sum_{\substack{(u,v)\in E \\ u\in S, v\in V\setminus S}} w(u,v),$$

where $w(u,v)$ denotes the weight of the edge (u,v). In the case of unweighted graphs, $w(u,v) = 1$ for all edges. The goal is to find a subset $S \subset V$ that maximizes $\mathrm{cut}(S)$. Figure 1 illustrates an example of a graph with 10 vertices and 14 edges, and how different partitions

lead to different cut sizes. As the figure shows, selecting an optimal partition significantly impacts the overall cut value. The max-cut problem is classified as NP-hard [1,2], which implies that no known polynomial-time algorithm can solve all instances optimally. Due to this computational difficulty, a wide variety of heuristic and metaheuristic approaches have been proposed to find high-quality approximate solutions efficiently. The problem has practical significance in many areas [3], such as VLSI design, statistical physics (e.g., the Ising model [4]), network design, and clustering tasks in machine learning.

Notable techniques include simulated annealing [5–7], genetic algorithms (GAs) [8–10], harmony search (HS) [11], and tabu search (TS) [6,12,13]. In addition to these methods, other metaheuristic frameworks such as scatter search [14], estimation of distribution algorithms (EDAs) [15], particle swarm optimization combined with EDAs [16], and ant colony optimization [17] have been extensively studied for solving the max-cut problem. These approaches have demonstrated effectiveness, particularly for large-scale instances where exact methods become computationally impractical. Beyond metaheuristics, neural network-based approaches have also been explored as alternative strategies for tackling the max-cut problem [18]. Notably, the discrete Hopfield neural network [19] leverages neural dynamics to iteratively refine solution quality. These machine learning-inspired techniques introduce a different paradigm compared to traditional metaheuristics, focusing on adaptive optimization processes.

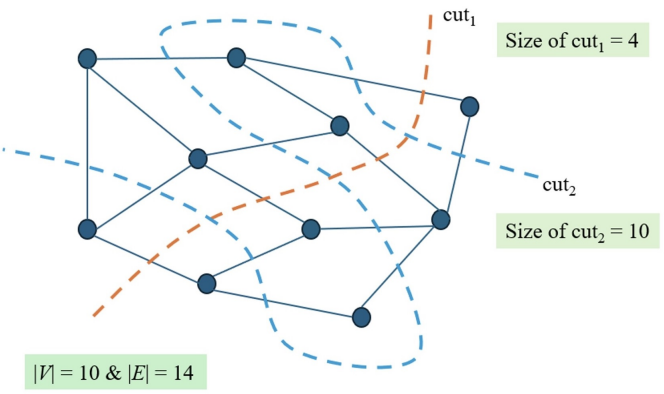

Figure 1. An illustrative example of the max-cut problem on a graph with 10 vertices and 14 edges. The figure demonstrates how different vertex partitions (cuts) result in different cut sizes. The objective of the max-cut problem is to find the partition that maximizes the number of edges crossing between the two subsets.

While metaheuristics and neural network-based methods provide effective heuristics for large-scale instances, exact and combinatorial optimization approaches remain crucial for obtaining optimal solutions or strong theoretical bounds. Research in this area has examined branch-and-bound techniques [20] and exact algorithms, particularly in sparse graphs [21]. Moreover, semidefinite programming relaxations [20,22,23] have been widely applied to derive strong upper bounds. Notably, rank-two relaxation heuristics [24] and polyhedral cut-and-price methods [23] have been proposed to improve computational efficiency.

Among the various metaheuristic approaches, recent studies have particularly highlighted the effectiveness of the harmony search (HS) algorithm in solving the max-cut problem. A study by Kim et al. [11] demonstrated that the HS algorithm outperforms GAs in solving the max-cut problem. One of the key features of HS is its mechanism for generating initial solutions in a way that avoids redundancy, preventing the algorithm from repeatedly exploring the same solutions. This characteristic is similar to the strategy employed by TS, which avoids revisiting previously explored solutions by maintaining a memory structure known as the tabu list. In our interpretation, this feature enables HS to

utilize the search space more effectively and potentially improve the quality of solutions. In contrast, while GA maintains diversity among candidate solutions, it may still suffer from repeatedly exploring similar regions in the solution space, which can lead to inefficiencies. To address this, we propose a new approach to enhance GA performance by incorporating key aspects of HS's search strategy.

In this study, we introduce a novel hybrid algorithm that combines GA and TS to improve GA's performance and introduce a search behavior similar to that of HS. By integrating TS's exploration constraint mechanisms into GA, we aim to mitigate the issue of redundant solution exploration and improve the overall efficiency of the search process. In addition, we compare the performance of the proposed algorithm with that of conventional GA and HS to demonstrate its effectiveness. Through this comparison, we try to establish that the combination of GA and TS not only outperforms GA alone, but also shares similarities with HS in terms of search behavior.

This study makes three key contributions. First, we propose a novel GATS algorithm that incorporates the constraints of TS to improve the search efficiency of GA and prevent repetitive exploration of solutions. Second, we empirically validate that GATS outperforms conventional GA in solving the max-cut problem while maintaining competitive performance with HS. Third, based on previous research showing HS's superiority over GA, we analyze how GATS exhibits similar search behavior to HS and discuss the implications of this within the broader context of recent trends in metaheuristic algorithms.

The structure of the paper is as follows. Section 2 reviews recent research, discussing existing algorithms for the max-cut problem and recent research trends. In Section 3, we introduce the genetic algorithms used in this study. Section 4 details the proposed GATS algorithm, including its design and integration strategy. Section 5 presents experimental results, comparing the performance of GATS with that of conventional GA and HS. Finally, Section 6 provides a summary of the main findings and outlines potential directions for future work.

2. Recent Work

Over the past decade, extensive research has been conducted on the max-cut problem, leading to the development of various algorithmic approaches, ranging from classical optimization techniques to advanced machine learning-based methods.

One prominent line of research focuses on classical optimization approaches. The Goemans–Williamson algorithm [25], based on semidefinite programming (SDP), remains one of the most influential methods for the max-cut problem. Building on this, Rodriguez-Fernandez et al. [26] explored whether replacing the random vector selection in this algorithm with clustering techniques such as k-means and k-medoids could yield better graph partitions. Mohades and Kahaei [27] took a different approach by reformulating the max-cut problem as an unconstrained optimization problem on a Riemannian manifold and applying a Riemannian gradient-based optimization algorithm. Meanwhile, Chen et al. [28] analyzed the limitations of local algorithms in large random hypergraphs, demonstrating that such methods struggle to find near-optimal max-cuts due to the overlap gap property. More recently, Kadhim and Al-jilawi [29] compared numerical SDP solvers—including bundle, interior point, and augmented Lagrangian methods—and reported that the bundle method achieved the fastest convergence on large benchmark instances.

Another significant research direction has focused on heuristic and metaheuristic methods for efficiently solving the max-cut problem. Wu et al. [30] introduced a hybrid evolutionary algorithm incorporating tabu search with a distance-and-quality-based solution combination operator and neighborhood-based search strategies. Lin and Zhu [31] proposed a memetic algorithm that integrates local search, a novel crossover operator,

and a population update mechanism designed to balance solution quality and diversity. Similarly, Zeng et al. [32] improved upon this approach by employing edge-state learning instead of vertex-position-based methods to enhance solution quality.

Various population-based heuristics have also been explored. Kim et al. [11] compared the harmony search algorithm with genetic algorithms, concluding that the former achieved superior results with fewer parameters and greater efficiency. In a related study, Kim et al. [33] proposed an incremental genetic algorithm that gradually expanded subproblems by adding edges or vertices in a structured manner. Soares and Araújo [34] further extended the genetic algorithm approach by integrating tabu search and optimality cuts, demonstrating their effectiveness in enhancing solution quality. Wang et al. [35] took a different perspective by classifying and refining greedy heuristics. Their relation tree-based framework categorized algorithms into vertex-oriented Prim-class approaches (e.g., Sahni–Gonzalez [36] variants) and edge-oriented Kruskal-class methods (e.g., edge contraction and its refinements). Chen et al. [37] introduced a hybrid binary artificial bee colony algorithm that combined surjection mapping with local search to improve solution efficiency. Šević et al. [38] introduced a learning-based fixed-set search (FSS) metaheuristic, which extracts common substructures from high-quality solutions and reuses them to guide new solution construction, outperforming traditional GRASP-based methods.

Recently, machine learning-based approaches have gained traction in solving the max-cut problem. Gu and Yang [18] pioneered the use of a pointer network-based model trained with supervised learning to tackle the problem. They later extended this work [39] by incorporating reinforcement learning strategies, making the approach more effective for large-scale instances. Yao et al. [40] investigated the performance of graph neural networks (GNNs) on the max-cut problem, showing that GNNs performed comparably to semidefinite relaxation but were outperformed by extremal optimization. Jin and Liu [41] proposed a physics-inspired method that formulated the max-cut problem as an Ising system, using the Hamiltonian of this system as a loss function to train a recurrent neural network for approximating the ground-state transition distribution. Garmendia et al. [42] further examined neural methods in combinatorial optimization, highlighting both the generalization potential of neural models trained on diverse graphs and the computational limitations that still prevent them from matching the reliability of traditional solvers.

Additionally, Gao [43] proposed a randomized fuzzy logic-based algorithm for computing local max-cuts in RP time, demonstrating significantly faster performance compared to IBM CPLEX solvers, especially on signed graphs with non-PSD Laplacians. Islam et al. [44] also developed a Harris Hawk Optimization (HHO) variant for max-cut, with additional operators like crossover, mutation, and repair to enhance both exploration and exploitation capabilities.

Finally, several recent studies have focused on experimental comparisons of different max-cut algorithms. Mirka and Williamson [45] conducted a comparative study of max-cut approximation algorithms, finding that spectral methods were significantly faster while often achieving results comparable to those of semidefinite programming. In addition, Kang et al. [46] demonstrated a neuromorphic hardware-based spiking Boltzmann machine that solves max-cut problems using probabilistic sampling and parallelism in phase-change memory circuits.

More recently, quantum computing-based approaches have been proposed as a promising direction for solving the max-cut problem. The quantum approximate optimization algorithm (QAOA) [47] utilizes quantum circuits to approximate near-optimal cuts and has shown potential in benchmark settings. Hybrid methods combining QAOA with classical pre- and post-processing have also demonstrated scalability on large graphs [48]. Furthermore, quantum GAs that integrate Grover search with divide-and-conquer evolutionary

strategies have been introduced and shown to outperform classical heuristics in specific instances [49]. These advances illustrate the growing relevance of quantum-enhanced optimization techniques in combinatorial problems like max-cut. These developments suggest that quantum-enhanced algorithms may become increasingly viable for tackling NP-hard problems like max-cut, especially as quantum hardware and software continue to evolve.

These diverse research efforts illustrate the continuous advancements in max-cut problem solving, with developments spanning classical optimization techniques, heuristic and metaheuristic approaches, emerging machine learning-driven methodologies, and now, quantum computing paradigms. These collective advances underline a continued effort to improve both the efficiency and exploratory capability of max-cut solvers. However, population-based heuristics such as GAs still suffer from redundant exploration. This motivates our proposed method, GATS (genetic algorithm with tabu search), which introduces solution-level tabu constraints to prevent revisiting previously explored solutions, inspired by mechanisms used in both TS and HS. As demonstrated in subsequent sections, GATS offers improved performance while maintaining computational efficiency.

3. Genetic Algorithms

Genetic algorithms (GAs) [50], as a representative technique of evolutionary computation [51], are widely used in combinatorial optimization problems such as graph partitioning [52–54]. They have also been extensively applied to the max-cut problem [8–11]. In this section, we introduce the GA used by Seo et al. [10] and propose an improved GA.

3.1. Generational Model

We now summarize the approach to the max-cut problem proposed by Seo et al. [10], where they consider two chromosome-encoding strategies for representing solutions to the max-cut problem:

- Vertex-based encoding: Each chromosome is a binary string of length $|V|$, where each gene corresponds to a vertex in the graph $G = (V, E)$. A gene value of 0 indicates that the corresponding vertex belongs to subset S, and 1 indicates it belongs to $V \setminus S$. This encoding allows for simple and intuitive computation of the cut size by counting the number of edges crossing between the two subsets.
- Edge-set encoding based on a spanning tree: As proposed by Seo et al. [10], this encoding represents each solution as a binary string of length $|V| - 1$, corresponding to a subset of edges from a predefined spanning tree of the graph. Each gene indicates whether the corresponding edge in the tree is included in the cut set. This representation enables structured exploration of the cut space by leveraging the properties of spanning trees.

These encoding methods serve as the basis for the genetic algorithms described in the following sections.

Seo et al. utilized a generational GA with the framework illustrated in Algorithm 1 and parameter settings detailed in Table 1. The algorithm employs tournament selection to choose candidate chromosomes, followed by one-point crossover and random bit-wise mutation. For implementation, Seo et al. [10] used Open Beagle [55] in combination with the Boost Graph Library (http://boost.org). Each execution of the genetic algorithm generates approximately 10,000 candidate solutions.

Table 1. Configuration of the generational genetic algorithm as described by Seo et al. [10].

Parameter	Value
Encoding	Vertex- or edge-set based
Length of run, T	100 generations
Population size, N	100
Crossover rate, p_c	0.9
Mutation rate, p_m	0.1
Tournament size, t	7
Replacement	A generational model where offspring with higher fitness replace their parents
Elitism	An elitism mechanism that retains superior individuals by replacing weaker ones

Algorithm 1 Pseudo-code of the generational genetic algorithm proposed by Seo et al. [10]

Randomly generate N solutions s_i, $i = 1, 2, \ldots, N$
repeat
  for $k = 1$ to $N/2$ **do**
    Tournament selection of parents s_{k_1} and s_{k_2} from the population
    $(o_{2k-1}, o_{2k}) \leftarrow$ one-point-crossover(s_{k_1}, s_{k_2}) with probability p_c
    $o_{2k-1} \leftarrow$ bit-wise mutation of o_{2k-1} with probability p_m
    $o_{2k} \leftarrow$ bit-wise mutation of o_{2k} with probability p_m
  end for
  Replace the population with offspring $o_1, o_2, \ldots, o_N$, keeping the fittest individuals
until when the predefined number T of generations is completed
return the most optimal solution discovered

3.2. Steady-State Model

We developed a new GA aimed at improving upon the approach described in Section 3.1, utilizing a steady-state GA framework, as illustrated in Algorithm 2. Each chromosome represents a partition $(S, V \setminus S)$ of a given graph $G = (V, E)$ and consists of $|V|$ genes, where each gene corresponds to a vertex in G. A gene is assigned a value of 0 if the corresponding vertex belongs to S, and 1 otherwise. This encoding method follows the vertex-based encoding strategy used in [10]. Each solution (chromosome) in our algorithm is represented by a binary vector:

$$\mathbf{x} = (x_1, x_2, \ldots, x_{|V|}) \in \{0, 1\}^{|V|}$$

where $x_i = 0$ indicates that vertex v_i belongs to subset S, and $x_i = 1$ indicates membership in $V \setminus S$. The fitness of a solution $\mathbf{x}$ is evaluated by the total number of edges crossing the partition, formally defined as

$$f(\mathbf{x}) = \sum_{(u,v) \in E} \delta(x_u \neq x_v)$$

where $\delta(\cdot)$ is the indicator function, returning 1 if the edge (u, v) connects two nodes in different subsets, and 0 otherwise. This formulation directly corresponds to the cut size of the solution.

The crossover operator used is uniform crossover, which creates offspring by randomly selecting each gene from one of the two parent chromosomes with equal probability. Mutation is applied bit-wise with a fixed probability p_m, flipping each gene independently. The algorithm starts by initializing a population of $N = 50$ randomly generated chromo-

somes. The fitness of each chromosome is determined by its cut size, and parent selection is performed randomly. A crossover operation generates two offspring by recombining genetic material from two parent chromosomes. We employ uniform crossover [56] and apply random bit-wise mutation. Once the offspring are generated, the GA replaces the least fit individual in the population with the fitter offspring. The algorithm runs for a fixed number of generations, denoted as T. Through experimentation, we found that setting $T = 5000$ results in execution times comparable to those of the method proposed by Seo et al. [10]. Over the course of each GA execution, approximately 10,000 candidate solutions are generated, following a Genitor-style replacement strategy [57].

Algorithm 2 Pseudo-code of our steady-state genetic algorithm

```
Randomly generate N solutions s_i, i = 1, 2, ..., N
repeat
    Randomly choose parents p_1 and p_2 from the population
    (o_1, o_2) ← uniform-crossover(p_1, p_2)
    o_1 ← bit-wise mutation of o_1 with probability p_m
    o_2 ← bit-wise mutation of o_2 with probability p_m
    o ← fitter of o_1 and o_2
    Replace the least fit member of the entire population with offspring o
until when the predefined number T of generations is completed
return the most optimal solution discovered
```

4. Proposed Algorithm

4.1. Overview of the Proposed Algorithm

Tabu search (TS), introduced by Glover [58], is a metaheuristic originally developed for operations research. It explores the solution space by iteratively modifying a single current solution. In each iteration, neighboring solutions are generated and evaluated. The best candidate not marked as tabu—or one that meets an aspiration criterion—is selected as the next current solution. The tabu list prevents cycling by restricting recently visited solutions. This process continues until a stopping condition is met, typically when no improvement is observed over several iterations.

The proposed method, genetic algorithm with tabu search (GATS), aims to enhance the performance of standard genetic algorithms (GAs) by integrating tabu search (TS) to prevent redundant exploration. Unlike conventional GAs, which often revisit previously explored solutions, GATS incorporates a tabu mechanism to enhance search efficiency and preserve solution diversity.

The memory mechanism in tabu search can assist a GA in escaping local minima and progressing toward a near-optimal solution. For combining GA with TS, we regard the population of GA as a tabu list. In this case, a solution obtained by crossover and mutation of GA is accepted only if it is not in the population. If the same solution already exists in the population, the solution is discarded and another solution is generated by applying crossover and mutation again. In this manner, we can design GA for exploring the solution space beyond local optimality and the GA combined with TS is expected to show improved performance.

4.2. Implementation of the Proposed Algorithm

The GATS framework follows the standard GA structure but incorporates tabu constraints at key stages. To formally describe the tabu mechanism, let the tabu list be denoted by

$$\mathcal{L} \subset \{0,1\}^{|V|}$$

which stores recently accepted solutions. A newly generated candidate solution $\mathbf{x}_{\text{new}}$ is accepted into the population only if

$$\mathbf{x}_{\text{new}} \notin \mathcal{L}$$

After acceptance, the tabu list is updated to include $\mathbf{x}_{\text{new}}$, and the worst element is removed if $|\mathcal{L}|$ exceeds the predefined size $|L|$. This maintains a fixed-size short-term memory that promotes exploration by avoiding repeated evaluations of the same solutions.

The algorithm starts with an initial population of solutions, each representing a potential max-cut partition. Parent selection follows a steady-state GA approach, using uniform crossover and random bit-wise mutation. However, instead of allowing duplicate solutions, a tabu list is maintained to track previously explored solutions. If a newly generated solution exists in the tabu list, it is discarded, and the search continues. This ensures that the search space is explored more effectively, reducing premature convergence. To detect duplicate solutions, the current implementation uses full-solution comparison. Although effective, this can be computationally demanding for large graphs. In future work, we plan to adopt a hash-based representation to improve scalability and performance.

Algorithm 3 shows the pseudo-code of the proposed algorithm combining GA with TS. The framework of the algorithm is almost the same as that of the steady-state GA introduced in Section 3.2. The algorithm employs vertex-based binary encoding [10], as outlined in Section 3.2. It starts by generating a population of 50 random chromosomes. These chromosomes are evaluated by calculating their cut sizes, and a given number of them are used to form a tabu list. For each generation, two chromosomes are selected randomly, and two new offspring are created using uniform crossover and random bit-wise mutation. If the fitter offspring is not in the tabu list, the GA replaces the least fit member of the entire population with the fitter offspring. However, if the fitter offspring is in the tabu list, it is simply discarded and the next generation begins. Note that the generation count increases regardless of whether a newly generated offspring is accepted or rejected. This design ensures consistent iteration control but may result in generations where no population update occurs.

Algorithm 3 Pseudo-code of our genetic algorithm combined with tabu search

Randomly generate N solutions s_i, $i = 1, 2, \ldots, N$
Add the generated random $|L|$ solutions to tabu list L
repeat
  Randomly choose parents p_1 and p_2 from the population
  $(o_1, o_2) \leftarrow$ uniform-crossover(p_1, p_2)
  $o_1 \leftarrow$ bit-wise mutation of o_1 with probability p_m
  $o_2 \leftarrow$ bit-wise mutation of o_2 with probability p_m
  $o \leftarrow$ fitter of o_1 and o_2
  if offspring $o \notin L$ **then**
    $w \leftarrow$ the least fit member of the population
    Replace w with o in the population
    if $w \in L$ **then**
      Remove w from the tabu list L
    else
      Remove the worst one from the tabu list L
    end if
    Add offspring o to the tabu list L
  end if
until when the predefined number T of generations is completed
return the most optimal solution discovered

4.3. Comparison with Traditional GA-TS Hybrids

A key distinction between the proposed GATS approach and existing hybridizations of GA with TS lies in how the tabu list is managed. While some recent studies have proposed hybrid GA-TS algorithms for the max-cut problem, they typically apply tabu mechanisms at the component or move level, such as tracking vertex-level changes. In contrast, our approach employs solution-level memory constraints, which, to the best of our knowledge, have not been extensively explored in the recent literature. Traditional GA-TS hybrids typically apply tabu constraints at the vertex level, tracking individual vertex movements to prevent cycling back to recently explored local optima [13,30,34]. While this method effectively improves local search performance, it does not prevent GA from generating redundant solutions at the global solution level, leading to inefficiencies in broader exploration.

In contrast, our proposed GATS (genetic algorithm with tabu search) integrates tabu search at the solution level, treating the GA population itself (or a part of it) as an implicit tabu list. This ensures that redundant solutions are eliminated at the crossover and mutation stages, thereby maintaining search diversity and improving overall efficiency. By restricting the reproduction of previously visited solutions, GATS achieves better exploration-exploitation balance compared to conventional GA-TS hybrids.

Although GATS is rooted in the GA framework, its solution-level memory mechanism strategically aligns it with the exploration behavior of harmony search (HS), rather than conventional GA-TS hybrids. Harmony search [59] maintains a memory structure that prevents redundant solution generation, ensuring continuous exploration of new regions. Similarly, GATS enforces solution-level constraints within GA's evolutionary process, promoting a more structured and effective search. This distinction is crucial, as it allows GATS to avoid unnecessary recomputation while maintaining population diversity, ultimately leading to superior performance in combinatorial optimization problems.

In sum, unlike traditional GA-TS hybrid methods that apply tabu constraints at a local level—such as tracking recent moves of individual vertices—the proposed GATS algorithm operates at the global solution level by treating the entire population (or a part of it) as an implicit tabu list. This global-level constraint reduces redundancy and sustains diversity, avoiding the need for additional memory structures required in conventional TS. As a result, GATS ensures a broader exploration of the solution space while avoiding the complexity of managing explicit memory structures as typically required in conventional TS mechanisms.

This population-as-memory mechanism distinguishes GATS as a novel hybridization framework, streamlining memory usage and enhancing search breadth. The effectiveness of this approach is further validated in Section 5 through experimental comparisons with HS.

This mechanism enables the algorithm to maintain diversity within the population by preventing the reintroduction of recently explored solutions, which could otherwise lead to premature convergence.

To better position the proposed GATS method within the context of existing approaches, we compare it with representative studies that have addressed solution duplication in GAs. Several prior studies have proposed methods to mitigate redundancy and improve diversity. For example, Mumford [60] introduced a replacement strategy in a steady-state multi-objective evolutionary algorithm that explicitly rejects phenotypically duplicate offspring from entering the population. Raidl and Hu [61] proposed a complete archive for binary-encoded GAs, utilizing a trie-based structure that stores all evaluated solutions and transforms revisited candidates into similar but unvisited ones using a controlled perturbation process. Additionally, Wang et al. [62] proposed a nonrevisiting genetic

algorithm based on a binary space partitioning (BSP) tree that not only records visited regions but also guides the search through multiple regions to avoid local optima.

Compared to these approaches, the GATS algorithm employs a fixed-length, solution-level tabu list that maintains a short-term memory of recently accepted solutions. This list is maintained separately from the population and is used to restrict the reinsertion of previously accepted individuals. Unlike trie-based or BSP-based global memory structures that require additional transformation logic or landscape modeling, our approach imposes a lightweight, recency-based avoidance strategy that integrates seamlessly with steady-state GA dynamics. Thus, while GATS shares the overarching goal of avoiding duplicate solutions and maintaining diversity, its implementation reflects a distinct balance between simplicity and effectiveness. In this sense, it represents a pragmatic hybridization of GA and TS principles, incorporating short-term memory without incurring significant computational overhead.

5. Experiments

All experiments were implemented in C and conducted on an Ubuntu 16.04.7 Linux system using a 2.8 GHz Intel i5 CPU. To ensure reproducibility, fixed random seeds were used in every run.

5.1. Test Graphs

We conducted experiments using 31 benchmark graphs commonly referenced in the literature [10,11,52,63]. These graphs represent different structural types and can be described as follows:

- G$n.p$: A randomly generated graph consisting of n vertices, where each edge is included with probability p. For example, G1000.01 denotes a random graph with 1,000 vertices and an edge probability of 0.01.
- U$n.d$: A geometric random graph with n vertices, where the expected degree of each vertex is d. As an example, U500.10 represents a geometric random graph with 500 vertices and an expected degree of 10.
- breg$n.b$: A 3-regular random graph with n vertices, constructed so that the optimal bisection size is close to b with high probability, which increases as n becomes larger.
- cat.n: A caterpillar graph with n vertices, where each node on the main spine is connected to six additional nodes. The variant rcat.n represents a modified caterpillar graph with n vertices, where each node along the spine has approximately $\sqrt{n}$ legs.
- grid$n.b$: A grid graph with n vertices, where the minimum cut size is b. The variant w-grid$n.b$ denotes a modified version with additional edges connecting opposite boundaries, forming a wrapped grid.

5.2. Determination of Tabu List Size

Selecting an appropriate size for the tabu list plays a critical role in guiding short-term search behavior [64], particularly when integrated with GAs. An overly small tabu list may allow repeated visits to previously explored solutions, while an excessively large list may overly restrict exploration and hinder convergence.

To determine the optimal size, we evaluated various tabu list sizes ranging from small to large values. The performance was assessed using the average of the percentage difference ratio, defined as $100 \times (best - output)/best$, where $best$ is the best solution found across all experiments for a given graph, and $output$ is the average solution obtained from 30 runs of the algorithm. Lower values indicate better performance.

As shown in Table 2, the smallest average difference ratio was achieved when the tabu list size was set to 5. This indicates that a size of 5 provides the most effective

balance between promoting exploration and avoiding redundant searches. In contrast, performance gradually deteriorates as the tabu list size increases beyond this point. This can be attributed to excessive restriction in solution acceptance, which may reduce diversity in the search space.

Based on these findings, we adopted a tabu list size of 5 as the default configuration in our proposed GATS algorithm.

Table 2. Impact of tabu list size on solution quality (average of percentage difference ratio).

Tabu List Size ($\|L\|$)	Average % Difference Ratio *
2	3.23
3	3.08
4	3.14
5	**3.07**
10	3.13
15	3.27
20	3.38
25	3.50
30	3.65
35	3.73
40	3.89
45	4.03
50	4.12
0: when a tabu list is not used	4.40

* The percentage difference ratio is calculated as $100 \times (best - output)/best$ (%), where $best$ is the highest cut value obtained among all runs, and $output$ is the average result of 30 runs. Lower values indicate better performance. Bold text highlights the best-performing result.

5.3. Comparison of Genetic Algorithms

In this subsection, we compare the performance of different GA variants applied to the max-cut problem, including the generational genetic algorithm (GGA) proposed by Seo et al. [10], our steady-state genetic algorithm (SGA), and our hybrid approach incorporating tabu search (GATS). To evaluate their effectiveness, we conducted experiments using benchmark graph instances, running each algorithm 30 times per instance and computing the average and standard deviation of the cut sizes. Table 3 compares their performance, where a multi-start algorithm (MSA) generates 10,000 random cuts and returns the best. The results indicate that SGA consistently outperforms GGA in our experiments, suggesting that the steady-state approach may be more effective than the generational model in maintaining solution quality under the given settings. Furthermore, GATS achieves superior results compared to SGA, highlighting the benefits of integrating tabu search to prevent redundant exploration and enhance search efficiency.

To confirm the statistical significance of these findings, we performed one-tailed t-tests comparing the results of GGA, SGA, and GATS. The p-values indicate that the improvements from GGA to SGA and from SGA to GATS are statistically significant, reinforcing the conclusion that our modifications lead to meaningful performance gains. Additionally, we analyzed computational efficiency and found that GATS maintains a runtime comparable to SGA while significantly improving solution quality. This suggests that incorporating tabu search does not introduce excessive computational overhead, making GATS a viable alternative to conventional GAs.

Table 3. Cut sizes and runtime comparison among four algorithms.

Graph	Multi-Start Algorithm			GGA [10]				SGA				GATS			
				Vertex		Edge Set									
	Ave	SD	Time *	Ave	SD	Ave	SD	Ave	SD	Time *	*p*-Value [1]	Ave	SD	Time *	*p*-Value [2]
G500.005	360.7	3.5	0.2	375.0	3.7	392.1	3.2	**519.9**	6.2	0.3	9.1×10^{-40}	519.0	6.7	0.3	3.0×10^{-1}
G500.01	680.0	6.4	0.3	700.1	4.5	708.1	5.7	926.4	10.1	0.3	4.3×10^{-40}	**931.8**	9.2	0.4	1.8×10^{-2}
G500.02	1274.0	7.5	0.4	1303.1	7.2	1295.7	7.5	1635.0	14.1	0.4	1.0×10^{-41}	**1643.0**	13.5	0.5	1.7×10^{-2}
G500.04	2695.3	9.9	0.7	2747.9	10.0	2722.2	12.1	3249.3	20.5	0.7	3.2×10^{-42}	**3263.1**	19.3	0.7	5.9×10^{-3}
G1000.005	1344.4	7.5	0.6	1373.4	8.4	1383.6	5.4	1873.7	12.9	0.7	3.4×10^{-48}	**1884.5**	13.9	0.7	2.0×10^{-3}
G1000.01	2669.1	11.1	0.9	2711.0	8.5	2702.0	7.8	3453.4	19.1	0.9	1.1×10^{-48}	**3472.0**	20.6	1.0	5.1×10^{-4}
G1000.02	5247.0	11.8	1.5	5307.4	10.9	5280.5	15.2	6367.4	28.6	1.4	1.2×10^{-47}	**6398.5**	26.8	1.5	7.5×10^{-5}
U500.05	702.9	4.9	0.3	597.2	3.6	606.3	3.8	847.0	6.9	0.3	2.2×10^{-46}	**850.2**	6.7	0.4	3.8×10^{-2}
U500.10	1259.7	5.7	0.4	1276.1	4.8	1283.8	6.5	1466.9	9.8	0.4	1.1×10^{-37}	**1474.0**	9.5	0.5	3.8×10^{-3}
U500.20	2381.4	7.9	0.7	2410.2	5.7	2407.4	5.4	2691.5	14.8	0.7	1.4×10^{-39}	**2705.6**	13.0	0.7	2.3×10^{-4}
U500.40	4538.4	7.1	1.1	4575.6	7.1	4563.2	8.2	5060.4	25.4	1.1	5.6×10^{-40}	**5077.8**	23.8	1.1	5.1×10^{-3}
U1000.05	1286.4	6.6	0.6	1309.0	7.6	1324.9	4.6	1601.9	8.2	0.7	5.8×10^{-46}	**1605.5**	9.4	0.7	6.2×10^{-2}
U1000.20	4834.1	10.8	1.4	4877.9	10.4	4871.1	10.6	5490.9	24.8	1.4	9.9×10^{-43}	**5516.3**	20.7	1.5	8.2×10^{-5}
U1000.40	9223.8	14.4	2.3	9292.5	12.2	9261.0	8.8	10,216.6	35.6	2.2	2.3×10^{-44}	**10,263.3**	36.3	2.3	1.1×10^{-5}
breg500.12	428.1	4.0	0.2	445.4	4.8	461.8	3.3	**635.1**	6.7	0.3	6.5×10^{-43}	634.1	5.6	0.3	2.6×10^{-1}
breg500.16	429.2	4.6	0.2	444.5	3.4	462.5	4.7	635.7	6.2	0.3	2.6×10^{-42}	**637.6**	6.9	0.3	1.3×10^{-1}
breg500.20	428.1	3.0	0.2	444.9	4.6	461.2	4.3	635.4	6.5	0.3	2.2×10^{-42}	**636.9**	6.4	0.3	1.9×10^{-1}
cat.352	210.6	2.5	0.1	224.1	2.9	242.7	2.8	**334.2**	3.1	0.2	5.3×10^{-42}	333.4	2.9	0.2	1.7×10^{-1}
cat.702	401.7	3.6	0.3	419.0	3.3	446.2	3.9	**666.3**	5.0	0.3	3.8×10^{-48}	666.2	5.0	0.4	4.6×10^{-1}
cat.1052	587.1	4.1	0.4	606.8	5.0	644.9	5.6	992.8	5.6	0.5	3.7×10^{-51}	**996.0**	6.1	0.6	2.1×10^{-2}
cat.5252	2764.9	11.8	2.3	2804.5	10.0	2891.2	10.2	4057.0	18.8	2.8	5.9×10^{-54}	**4073.4**	19.9	3.2	1.3×10^{-3}
rcat.134	89.0	2.1	0.1	99.4	1.5	106.1	1.5	**129.12**	1.6	0.1	1.3×10^{-32}	128.9	1.5	0.1	3.0×10^{-1}
rcat.554	321.8	4.1	0.2	339.0	3.1	360.2	2.5	**543.5**	2.3	0.2	7.0×10^{-54}	543.1	2.3	0.3	3.0×10^{-1}
rcat.994	555.8	3.7	0.4	577.9	5.2	611.3	4.7	974.2	3.3	0.5	6.1×10^{-56}	**976.4**	3.1	0.5	5.1×10^{-3}
rcat.5114	2694.0	9.5	2.1	2736.6	10.3	2817.6	9.7	4155.9	17.6	2.5	1.3×10^{-56}	**4198.7**	19.7	2.9	3.2×10^{-10}
grid100.10	116.9	2.7	0.1	145.1	2.6	146.8	1.9	153.4	6.4	0.1	3.9×10^{-06}	**161.79**	8.3	0.1	6.8×10^{-5}
grid500.21	536.0	4.0	0.3	582.1	4.6	594.5	5.4	803.4	14.4	0.3	6.7×10^{-36}	**815.6**	15.8	0.3	1.9×10^{-3}
grid1000.20	1050.1	5.3	0.5	1113.8	9.0	1133.2	7.0	1610.9	21.4	0.6	1.2×10^{-41}	**1621.1**	20.9	0.7	3.6×10^{-2}
w-grid100.20	128.4	3.4	0.1	130.1	2.4	133.7	1.8	164.3	7.0	0.1	5.3×10^{-21}	**173.0**	12.5	0.1	1.2×10^{-3}
w-grid500.42	559.4	3.6	0.3	556.9	4.3	571.7	6.0	828.1	16.5	0.3	8.1×10^{-37}	**835.4**	18.5	0.3	6.1×10^{-2}
w-grid1000.40	1087.9	7.0	0.5	1075.5	5.6	1096.3	5.1	1641.7	22.5	0.6	4.3×10^{-43}	**1653.7**	21.9	0.7	2.3×10^{-2}

Results from 30 runs. (Ave: average cut size. SD: standard deviation). * Average execution time in seconds using a 2.8 GHz Intel i5 CPU. [1] A one-tailed *t*-test was used to assess whether the generational GA [10] performs equivalently to the SGA. [2] A one-tailed *t*-test was used to evaluate whether SGA and GATS produce equivalent results. Bold text highlights the best-performing result in each row.

Overall, the comparison reveals a clear performance hierarchy among the tested methods, where GGA is outperformed by SGA, which in turn is outperformed by GATS. The steady-state approach in SGA contributes to better convergence, while the integration of TS in GATS further refines the search process by mitigating local optima entrapment. These results confirm that our proposed hybrid method effectively balances exploration and exploitation, leading to a more robust optimization strategy for the max-cut problem. Overall, the performance ranking of the methods can be summarized as follows: MSA $\ll$ GGA $\ll$ SGA $\ll$ GATS, where "A $\ll$ B" means that B significantly outperforms A.

5.4. Analysis of Crossover Operator Impact

To better understand the source of performance improvement in the proposed steady-state genetic algorithm (SGA), we conducted a decomposition analysis, comparing the contribution of the evolutionary scheme and the crossover operator separately. While GGA employs a generational model with one-point crossover, the proposed SGA uses a steady-state model with uniform crossover.

To isolate the effect of the crossover operator, we implemented a variant of SGA that adopts one-point crossover instead of uniform crossover. Figure 2 presents the comparative improvement rates across 31 benchmark instances: the yellow bars represent the improvement of SGA (with uniform crossover) over GGA, while the orange bars indicate the improvement achieved solely by switching from one-point to uniform crossover within the SGA framework.

As shown in the figure, uniform crossover contributes significantly to the overall performance gain in many instances. In particular, instances such as `prcart.994`, `pcart.1052`, and `rcat.554` show that more than one-fourth of the total improvement comes from the crossover operator alone. These results suggest that both the steady-state scheme and the choice of crossover operator play important and complementary roles in enhancing the effectiveness of the algorithm.

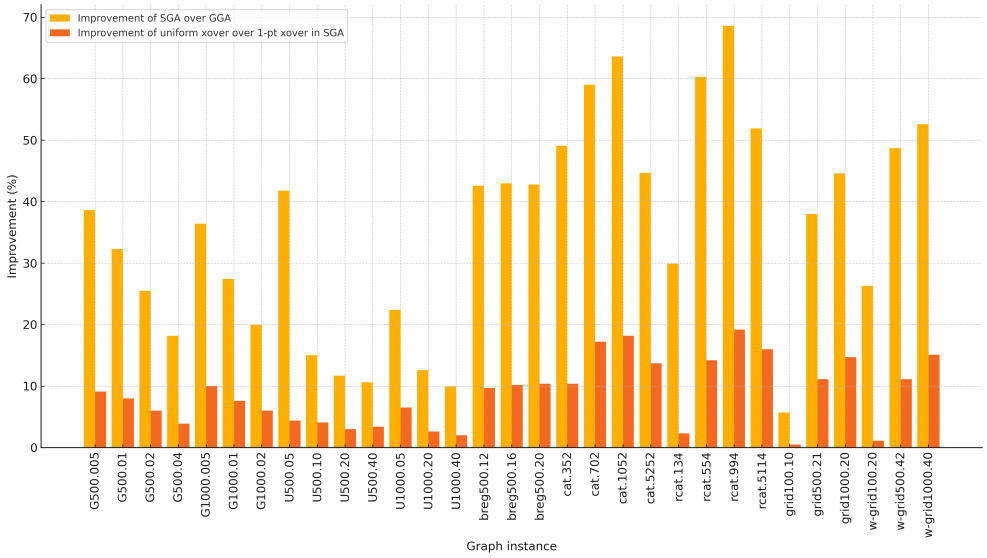

Figure 2. Comparative improvement analysis showing the contribution of steady-state evolution and uniform crossover. Yellow bars indicate the improvement of SGA over GGA, while orange bars indicate the performance gain due to the use of uniform crossover over one-point crossover in SGA.

5.5. Convergence Analysis

To further investigate the convergence behavior of the proposed GATS algorithm, we compared its performance against SGA over generations. Specifically, we measured the

average cut size of the entire population at each generation for two representative graph instances: `G1000.005` and `U1000.05`.

Figures 3 and 4 present the convergence curves for these instances. As shown in the plots, GATS not only achieves higher average cut sizes throughout the evolution process but also exhibits more stable convergence than SGA. These results support the claim that the integration of a solution-level tabu list in GATS not only improves population diversity but also contributes to more efficient and effective convergence.

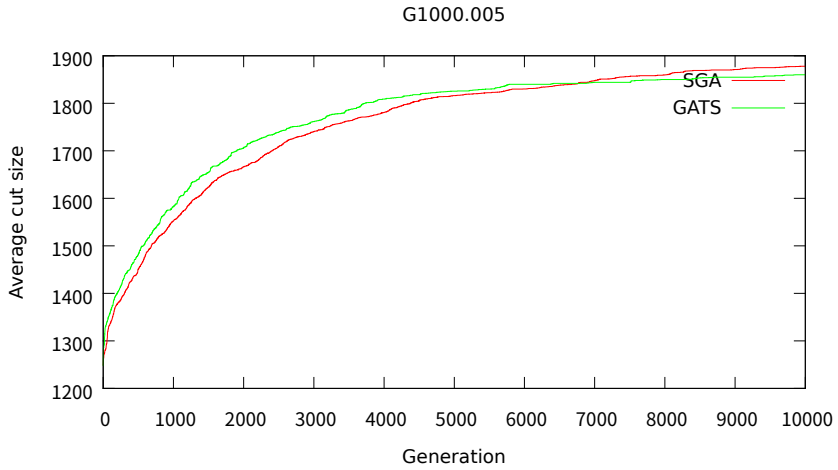

Figure 3. Convergence behavior on instance `G1000.005`: average cut size over generations.

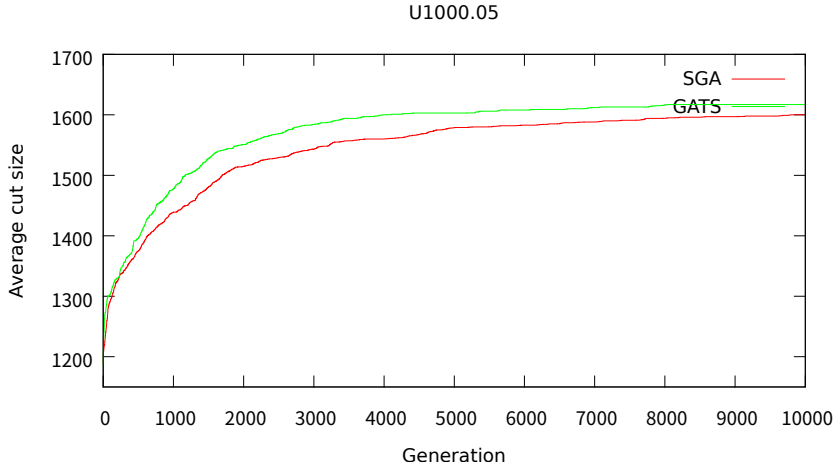

Figure 4. Convergence behavior on instance `U1000.05`: average cut size over generations.

5.6. Comparison with Harmony Search

In this subsection, we analyze the results presented in Table 4, which compare the performance of GATS and HS. The results indicate that the two algorithms exhibit generally similar performance, with only minor differences in most cases. The average cut sizes obtained by GATS and HS are comparable across various test graphs, and the standard deviations remain consistently close, suggesting that GATS effectively incorporates the advantages of TS while maintaining search behavior similar to HS.

However, certain test instances reveal notable performance differences between the two algorithms. In particular, HS significantly outperforms GATS in specific graphs such as `cat.5252` and `rcat.5114`. For these cases, HS achieves substantially larger cut sizes, indicating that its search strategy is more effective for these graph structures. On the other hand, GATS performs slightly better than HS in a few instances, such as `G1000.01` and `breg500.12`, though the differences in these cases are relatively small. The statistical

significance analysis further supports these observations. While the majority of test cases yield p-values greater than 0.1, suggesting that the differences between the two algorithms are not statistically significant, certain cases, including `cat.5252` and `rcat.5114`, exhibit extremely low p-values, confirming that HS significantly outperforms GATS for these specific graphs. Conversely, in cases where GATS shows marginally better results, such as `G1000.01`, the difference is relatively minor but still noticeable.

Another important aspect to consider is the execution time of the two algorithms. The results indicate that GATS and HS require nearly identical computation times, implying that the introduction of TS in GATS does not impose additional computational overhead. This suggests that while TS effectively prevents redundant solution exploration, it achieves this without compromising efficiency.

Table 4. Performance comparison of the proposed method and harmony search.

	GATS			HS			
Graph	**Ave**	**SD**	**Time ***	**Ave**	**SD**	**Time ***	***p*-Value [1]**
G500.005	519.0	6.7	0.3	**521.3**	6.6	0.3	9.3×10^{-2}
G500.01	931.8	9.2	0.4	**933.1**	11.6	0.4	3.2×10^{-1}
G500.02	**1643.0**	13.5	0.5	1642.1	16.2	0.5	4.1×10^{-1}
G500.04	**3263.1**	19.3	0.7	3259.3	19.5	0.7	2.3×10^{-1}
G1000.005	**1884.5**	13.9	0.7	1883.6	14.2	0.8	4.0×10^{-1}
G1000.01	**3472.0**	20.6	1.0	3460.7	21.3	1.0	2.1×10^{-2}
G1000.02	6398.5	26.8	1.5	**6402.5**	24.4	1.5	2.7×10^{-1}
U500.05	850.2	6.7	0.4	**852.2**	6.9	0.4	1.3×10^{-1}
U500.10	**1474.0**	9.5	0.5	1473.0	8.5	0.5	3.3×10^{-1}
U500.20	**2705.6**	13.0	0.7	2704.5	10.3	0.7	3.6×10^{-1}
U500.40	5077.8	23.8	1.1	**5086.3**	22.9	1.1	8.2×10^{-2}
U1000.05	**1605.5**	9.4	0.7	1604.4	9.1	0.8	3.2×10^{-1}
U1000.20	**5516.3**	20.7	1.5	5509.9	21.7	1.5	1.2×10^{-1}
U1000.40	**10,263.3**	36.3	2.3	10,251.5	36.0	2.3	1.1×10^{-1}
breg500.12	634.1	5.6	0.3	**636.5**	4.0	0.3	3.1×10^{-2}
breg500.16	637.6	6.9	0.3	**639.2**	6.0	0.3	1.7×10^{-1}
breg500.20	**636.9**	6.4	0.3	636.6	6.2	0.3	4.3×10^{-1}
cat.352	333.4	2.9	0.2	**335.2**	3.1	0.2	1.2×10^{-2}
cat.702	666.2	5.0	0.4	**666.9**	4.8	0.4	2.9×10^{-1}
cat.1052	996.0	6.1	0.6	**997.2**	4.7	0.6	2.0×10^{-1}
cat.5252	4073.4	19.9	3.2	**4309.5**	23.8	2.3	3.8×10^{-44}
rcat.134	128.9	1.5	0.1	**130.0**	1.4	0.1	2.4×10^{-3}
rcat.554	**543.1**	2.3	0.3	542.8	2.5	0.3	3.2×10^{-1}
rcat.994	976.4	3.1	0.5	**978.3**	3.4	0.5	1.4×10^{-2}
rcat.5114	4198.7	19.7	2.9	**4479.1**	18.4	2.0	3.0×10^{-52}
grid100.10	161.79	8.3	0.1	**164.3**	7.1	0.1	1.1×10^{-1}
grid500.21	**815.6**	15.8	0.3	811.0	16.1	0.3	1.3×10^{-1}
grid1000.20	1621.1	20.9	0.7	**1622.8**	25.7	0.7	3.9×10^{-1}
w-grid100.20	**173.0**	12.5	0.1	171.0	9.5	0.1	2.4×10^{-1}
w-grid500.42	835.4	18.5	0.3	**842.1**	23.2	0.3	1.1×10^{-1}
w-grid1000.40	**1653.7**	21.9	0.7	1651.8	23.7	0.7	3.7×10^{-1}

Results from 30 runs. (Ave: average cut size. SD: standard deviation). * Average execution time in seconds using a 2.8 GHz Intel i5 CPU. [1] A one-tailed t-test was used to assess whether GATS and HS yield equivalent results. Bold text highlights the best-performing result in each row.

In sum, the comparison between GATS and HS demonstrates that the two algorithms achieve similar performance, with HS exhibiting an advantage in certain graph instances and GATS performing slightly better in others. The negligible computational cost of integrating TS in GATS further reinforces its practical utility. These findings indicate that

GATS serves as an effective alternative to HS while maintaining the strengths of GA-based search strategies. Future research could further explore the underlying reasons why HS outperforms GATS in certain cases and investigate potential hybrid approaches that incorporate additional features of HS into GATS to enhance its overall performance.

The results from Section 5 collectively demonstrate that GATS not only enhances GA performance but also provides a competitive alternative to HS across a range of graph types.

To further illustrate the performance differences among the algorithms, we present a bar chart in Figure 5. This visualization complements the numerical results shown in Tables 3 and 4, allowing for a more intuitive comparison. As seen in the figure, the GATS and HS algorithms consistently outperform GGA and SGA across most benchmark instances, with HS showing a noticeable advantage in some specific cases (e.g., `cat.5252`, `rcat.5114`).

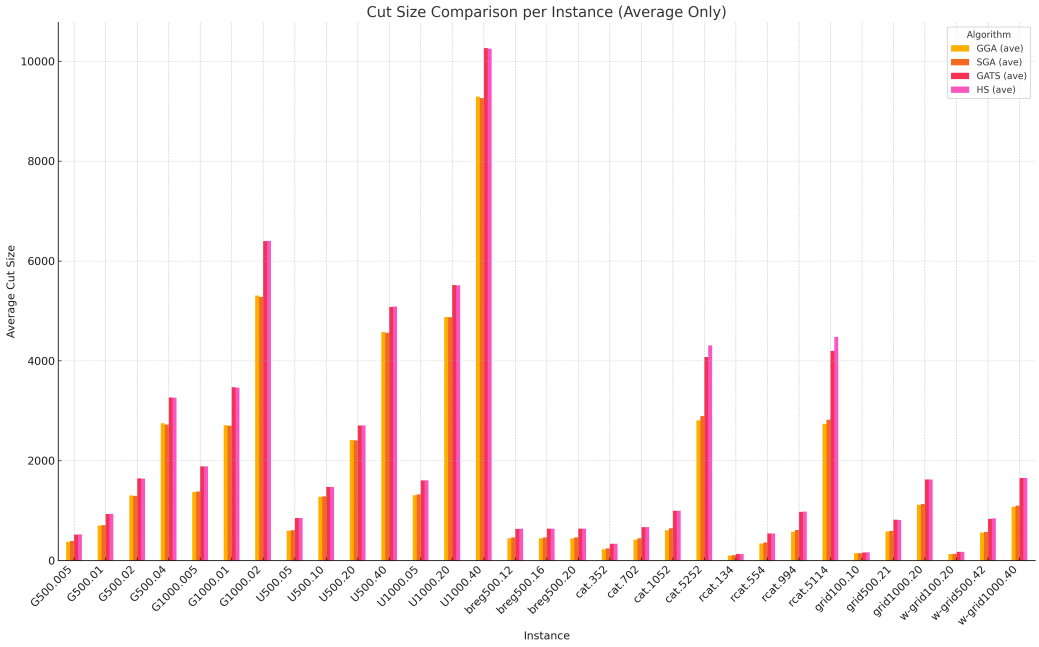

Figure 5. Bar chart comparing the average cut sizes achieved by four algorithms (GGA, SGA, GATS, and HS) across all benchmark graph instances. This visualization complements the tabulated results and illustrates the relative performance more clearly.

In particular, HS appears to perform better than GATS in instances such as `cat.5252` and `rcat.5114`. One possible reason is that HS generates new solutions by combining components from multiple existing solutions (i.e., a multi-parent mechanism), which may allow for more flexible exploration compared to the two-parent crossover in GA. This approach could be more effective for certain graph structures, such as tree-like graphs, where diverse component mixing helps escape local optima. These observations suggest that the recombination strategy in GATS may be less suited to specific topologies.

6. Conclusions and Future Work

This study proposed a hybrid genetic algorithm that incorporates tabu search (GATS) to improve the performance of GAs in solving the max-cut problem. By integrating TS, the proposed method effectively mitigates redundant exploration, leading to a more efficient search process. The experimental results confirm that GATS outperforms the standard GAs while exhibiting search behavior similar to harmony search (HS). This suggests that mitigating repetitive searches via tabu mechanisms contributes to more effective combinatorial optimization.

The findings of this study highlight the potential of combining GAs with TS to improve search efficiency. The results indicate that this approach can serve as an alternative to HS, which has been shown to be effective in previous studies. Furthermore, the similarities in search strategies between GATS and HS suggest that such hybridization may bridge the performance gap between various heuristic methods and provide a unified framework for efficient combinatorial optimization.

Unlike previous GA-TS hybrids, the proposed GATS method applies tabu constraints at the solution level, effectively balancing exploration and exploitation by discarding previously visited solutions at the generation stage. This novel mechanism contributes to improved performance without increasing computational complexity.

Future Research Directions

While the proposed method demonstrates promising performance improvements, several aspects can be explored in future research:

- Theoretical analysis of GATS efficiency: While the empirical results validate the effectiveness of GATS, a more rigorous theoretical analysis is required to explain why the method performs better than standard GAs. Future work could analyze the impact of tabu constraints on exploration efficiency and convergence properties using mathematical models.
- Diversity maintenance and search space coverage: The tabu list in GATS prevents redundant solutions, but its effect on search space coverage and genetic diversity has not been explicitly measured. Future studies could investigate the diversity preservation mechanisms of GATS using entropy-based or distribution-based metrics. Additionally, visualizing search trajectory patterns could provide further insights into how GATS explores the solution space differently compared to traditional GAs.
- Performance on different graph structures: While the proposed algorithm has been tested on a variety of benchmark graphs, further experiments on larger or structurally different graph types, such as real-world network graphs, could enhance our understanding of its generalizability. In particular, cases where HS significantly outperforms GATS, such as `cat.5252` and `rcat.5114`, require a deeper analysis to identify potential limitations of the current approach.
- Comparisons with other metaheuristic techniques: Future work could explore hybridizing GATS with other metaheuristic algorithms [65] beyond HS. Techniques such as simulated annealing or reinforcement learning-based search strategies could be incorporated to enhance adaptability and convergence efficiency. Evaluating how these hybrid approaches perform against state-of-the-art heuristics would provide valuable insights.
- Computational complexity and optimization: Although GATS does not introduce significant computational overhead, further optimization of the algorithm could improve its scalability for large-scale instances. Investigating more efficient tabu list management strategies or parallelized implementations could enhance runtime efficiency while maintaining solution quality. Additionally, future studies could explore combining GATS with reinforcement learning-guided selection mechanisms to dynamically adapt tabu constraints during the search.

By addressing these research directions, future work can further refine the effectiveness of hybrid GAs in solving combinatorial optimization problems and extend their applicability to broader domains.

Author Contributions: Conceptualization, Y.-H.K. and Y.Y.; methodology, Y.-H.K.; software, Y.-H.K.; validation, Y.Y.; formal analysis, Y.Y.; investigation, Z.W.G. and Y.Y.; resources, Y.-H.K.; data

curation, Y.-H.K.; writing—original draft preparation, Y.-H.K.; writing—review and editing, Z.W.G. and Y.Y.; visualization, Y.Y.; supervision, Z.W.G. and Y.Y.; project administration, Z.W.G. and Y.Y.; funding acquisition, Y.-H.K. and Z.W.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry & Energy, Republic of Korea (RS-2024-00442817) and also supported by the Korea Institute of Marine Science & Technology Promotion (KIMST) funded by the Ministry of Oceans and Fisheries, Korea (RS-2022-KS221629).

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The present research has been conducted by the Research Grant of Kwangwoon University in 2024.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GA	Genetic algorithm
HS	Harmony search
TS	Tabu search
EDA	Estimation of distribution algorithms
GNN	Graph neural network
MSA	Multi-start algorithm
GGA	Generational GA
SGA	Steady-state GA
GATS	GA combined with TS

References

1. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Plenum Press: New York, NY, USA, 1972; pp. 85–103.
2. Garey, M.R.; Johnson, D.S.; Stockmeyer, L.J. Some simplified NP-complete graph problems. *Theor. Comput. Sci.* **1976**, *1*, 237–267. [CrossRef]
3. Barahona, F.; Grotschel, M.; Junger, M.; Reinelt, G. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.* **1988**, *36*, 493–513. [CrossRef]
4. Anderson, C.A.; Jones, K.F.; Ryan, J. A Two-Dimensional Genetic Algorithm for The Ising Problem. *Complex Syst.* **1991**, *5*, 327–333.
5. Homer, S.; Peinado, M. Design and performance of parallel and distributed approximation algorithms for maxcut. *J. Parallel Distrib. Comput.* **1997**, *46*, 48–61. [CrossRef]
6. Arráiz, E.; Olivo, O. Competitive simulated annealing and tabu search algorithms for the max-cut problem. In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montréal, QC, Canada, 8–12 July 2009; pp. 1797–1798. [CrossRef]
7. Myklebust, T. Solving maximum cut problems by simulated annealing. *arXiv* **2015**, arXiv:1503.05774.
8. Kim, S.H.; Kim, Y.H.; Moon, B.R. A hybrid genetic algorithm for the MAX CUT problem. In Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 7–11 July 2001; pp. 416–423.
9. Wu, Q.; Hao, J.K. A memetic approach for the Max-Cut problem. In Proceedings of the PPSN 2012—Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, 1–5 September 2012; Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M., Eds.; LNCS; Springer: Heidelberg/Berlin, Germany, 2012; Volume 7492, pp. 297–306. [CrossRef]
10. Seo, K.; Hyun, S.; Kim, Y.H. An edge-set representation based on a spanning tree for searching cut space. *IEEE Trans. Evol. Comput.* **2015**, *19*, 465–473. [CrossRef]
11. Kim, Y.; Yoon, Y.; Geem, Z. A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm Evol. Comput.* **2019**, *44*, 130–135. [CrossRef]
12. Palubeckis, G. Application of multistart tabu search to the MaxCut problem. *Inform. Technol. Control* **2004**, *2*, 29–35.

13. Kochenberger, G.; Hao, J.; Lü, Z.; Wang, H.; Glover, F. Solving large scale Max Cut problems via tabu search. *J. Heuristics* **2013**, *19*, 565–571. [CrossRef]
14. Martí, R.; Duarte, A.; Laguna, M. Advanced scatter search for the Max-Cut problem. *INFORMS J. Comput.* **2009**, *21*, 26–38. [CrossRef]
15. de Sousa, S.; Haxhimusa, Y.; Kropatsch, W. Estimation of distribution algorithm for the max-cut problem. In Proceedings of the Graph-Based Representations in Pattern Recognition: 9th IAPR-TC-15 International Workshop, GbRPR 2013, Vienna, Austria, 15–17 May 2013; Proceedings 9; Springer: Berlin/Heidelberg, Germany, 2013; pp. 244–253. [CrossRef]
16. Lin, G.; Guan, J. An Integrated Method Based on PSO and EDA for the Max-Cut Problem. *Comput. Intell. Neurosci.* **2016**, *2016*, 3420671. [CrossRef] [PubMed]
17. Gao, L.; Zeng, Y.; Dong, A. An ant colony algorithm for solving Max-cut problem. *Prog. Nat. Sci.* **2008**, *18*, 1173–1178. [CrossRef]
18. Gu, S.; Yang, Y. A pointer network based deep learning algorithm for the max-cut problem. In Proceedings of the Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, 13–16 December 2018; Proceedings, Part I 25; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 238–248. [CrossRef]
19. Wang, J. An improved discrete Hopfield neural network for Max-Cut problems. *Neurocomputing* **2006**, *69*, 1665–1669. [CrossRef]
20. Rendl, F.; Rinaldi, G.; Wiegele, A. A branch and bound algorithm for Max-Cut based on combining semidefinite and polyhedral relaxations. In Proceedings of the IPCO—Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Ithaca, NY, USA, 25–27 June 2007; Fischetti, M., Williamson, D.P., Eds.; LNCS; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4513, pp. 295–309. [CrossRef]
21. Crocea, F.; Kaminskib, M.; Paschosc, V. An exact algorithm for MAX-CUT in sparse graphs. *Oper. Res. Lett.* **2007**, *35*, 403–408. [CrossRef]
22. Krislock, N.; Malick, J.; Rouoin, F. Improved semidefinite bounding procedure for solving Max-Cut problems to optimality. *Math. Program.* **2014**, *143*, 61–86. [CrossRef]
23. Krishnan, K.; Mitchell, J. A semidefinite programming based polyhedral cut and price approach for the maxcut problem. *Comput. Optim. Appl.* **2006**, *33*, 51–71. [CrossRef]
24. Burer, S.; Monteiro, R.; Zhang, Y. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM J. Optim.* **2002**, *12*, 503–521. [CrossRef]
25. Goemans, M.X.; Williamson, D.P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **1995**, *42*, 1115–1145. [CrossRef]
26. Rodriguez-Fernandez, A.; Gonzalez-Torres, B.; Menchaca-Mendez, R.; Stadler, P. Clustering improves the Goemans–Williamson approximation for the max-cut problem. *Computation* **2020**, *8*, 75. [CrossRef]
27. Mohades, M.; Kahaei, M. An Efficient Riemannian Gradient Based Algorithm for Max-Cut Problems. *IEEE Trans. Circuits Syst. II: Express Briefs* **2021**, *69*, 1882–1886. [CrossRef]
28. Chen, W.K.; Gamarnik, D.; Panchenko, D.; Rahman, M. Suboptimality of local algorithms for a class of max-cut problems. *Ann. Probab.* **2019**, *47*, 1587–1618. [CrossRef]
29. Kadhim, F.J.; Al-jilawi, A.S. A Hybrid Approximate Algorithms of Numerical Optimization for Solving the Max-Cut Problem. *arXiv* **2024**. [CrossRef]
30. Wu, Q.; Wang, Y.; Lü, Z. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Appl. Soft Comput.* **2015**, *34*, 827–837. [CrossRef]
31. Lin, G.; Zhu, W.X. A memetic algorithm for the max-cut problem. *Int. J. Comput. Sci. Manag.* **2015**, *10*, 67–544. [CrossRef]
32. Zeng, Z.; Yu, X.; Wu, Q.; Wang, Y.; Zhou, Z. A memetic algorithm based on edge-state learning for max-cut. *Expert Syst. Appl.* **2022**, *209*, 118077. [CrossRef]
33. Kim, J.; Yoon, Y.; Moon, B. Solving maximum cut problem with an incremental genetic algorithm. In Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, Denver, CO, USA, 20–24 July 2016; ACM: New York, NY, USA, 2016; pp. 49–50. [CrossRef]
34. Soares, P.; Araújo, C. Genetic Algorithms with Optimality Cuts to the Max-Cut Problem. In Proceedings of the Brazilian Conference on Intelligent Systems, Belo Horizonte, Brazil, 25–29 September 2023; Springer: Cham, Switzerland, 2023; pp. 17–32. [CrossRef]
35. Wang, J.; Wu, C.; Zuo, F. More on greedy construction heuristics for the MAX-CUT problem. *arXiv* **2023**, arXiv:2312.10895.
36. Sahni, S.; Gonzalez, T. P-complete approximation problems. *J. ACM* **1976**, *23*, 555–565. [CrossRef]
37. Chen, X.; Lin, G.; Xu, M. Applying a binary artificial bee colony algorithm to the max-cut problem. In Proceedings of the 2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Suzhou, China, 19–21 October 2019; IEEE: New York, NY, USA, 2019; pp. 1–4. [CrossRef]
38. Šević, I.; Jovanovic, R.; Urošević, D.; Davidović, T. Fixed Set Search Applied to the Max-Cut Problem. In Proceedings of the 8th IEEE Energy Conference (ENERGYCON), Doha, Qatar, 4–7 March 2024; pp. 1–6. [CrossRef]

39. Gu, S.; Yang, Y. A deep learning algorithm for the max-cut problem based on pointer network structure with supervised learning and reinforcement learning strategies. *Mathematics* **2020**, *8*, 298. [CrossRef]
40. Yao, W.; Bandeira, A.; Villar, S. Experimental performance of graph neural networks on random instances of max-cut. In Proceedings of the Wavelets and Sparsity XVIII, San Diego, CA, USA, 13–15 August 2019; SPIE: Bellingham, WA, USA, 2019; Volume 11138, pp. 242–251. [CrossRef]
41. Jin, A.; Liu, X. A Fast Machine Learning Algorithm for the MaxCut Problem. In Proceedings of the 2023 IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, USA, 6–8 October 2023; IEEE: New York, NY, USA, 2023; pp. 1–5. [CrossRef]
42. Garmendia, A.I.; Ceberio, J.; Mendiburu, A. Exploring the Capabilities and Limitations of Neural Methods in the Maximum Cut. In Proceedings of the Advances in Artificial Intelligence, CAEPIA 2024, A Coruña, Spain, 19–21 June 2024; Alonso-Betanzos, A., Guijarro-Berdiñas, B., Bolón-Canedo, V., Hernández-Pereira, E., Fontenla-Romero, O., Camacho, D., Rabuñal, J.R., Ojeda-Aciego, M., Medina, J., Riquelme, J.C., et al., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2024; Volume 14640. [CrossRef]
43. Gao, L. Finding local Max-Cut in graphs in randomized polynomial time. *Soft Comput.* **2024**, *28*, 3029–3048. [CrossRef]
44. Islam, M.R.; Islam, M.S.; Boni, P.K.; Sarker, A.S.; Anam, M.A. Solving the maximum cut problem using Harris Hawk Optimization algorithm. *PLoS ONE* **2025**, *19*, e0315842. [CrossRef]
45. Mirka, R.; Williamson, D. An experimental evaluation of semidefinite programming and spectral algorithms for max cut. *ACM J. Exp. Algorithmics* **2023**, *28*, 1–18. [CrossRef]
46. Kang, Y.G.; Ishii, M.; Park, J.; Shin, U.; Jang, S.; Yoon, S.; Kim, M.; Okazaki, A.; Ito, M.; Nomura, A.; et al. Solving Max-Cut Problem Using Spiking Boltzmann Machine Based on Neuromorphic Hardware with Phase Change Memory. *Adv. Sci.* **2024**, *11*, 2406433. [CrossRef]
47. Farhi, E.; Goldstone, J.; Gutmann, S. A Quantum Approximate Optimization Algorithm. *arXiv* **2014**, arXiv:1411.4028.
48. Angone, A.; Liu, X.; Shaydulin, R.; Safro, I. Hybrid Quantum-Classical Multilevel Approach for Maximum Cuts on Graphs. *arXiv* **2023**, arXiv:2309.08815.
49. Viana, P.A.; de Paula Neto, F.M. A Quantum Genetic Algorithm Framework for the MaxCut Problem. *arXiv* **2024**, arXiv:2501.01058.
50. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley: Boston, MA, USA, 1989.
51. Kim, Y.H.; Caraffini, F. Preface to "Swarm and Evolutionary Computation—Bridging Theory and Practice". *Mathematics* **2023**, *11*, 1209. [CrossRef]
52. Kim, Y.H.; Moon, B.R. Lock-gain based graph partitioning. *J. Heuristics* **2004**, *10*, 37–57. [CrossRef]
53. Moraglio, A.; Kim, Y.H.; Yoon, Y.; Moon, B.R. Geometric Crossovers for Multiway Graph Partitioning. *Evol. Comput.* **2007**, *15*, 445–474. [CrossRef]
54. Kim, J.; Hwang, I.; Kim, Y.H.; Moon, B.R. Genetic approaches for graph partitioning: A survey. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, 12–16 July 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 473–480. [CrossRef]
55. Gagné, C.; Parizeau, M. Genericity in evolutionary computation software tools: Principles and case study. *Int. J. Artif. Intell. Tools* **2006**, *15*, 173–194. [CrossRef]
56. Syswerda, G. Uniform Crossover in Genetic Algorithms. In Proceedings of the Third International Conference on Genetic Algorithms, Fairfax, WV, USA, 1 June 1989; pp. 2–9.
57. Whitley, D.; Kauth, J. GENITOR: A different genetic algorithm. In Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Saint Paul, MN, USA, 21–26 August 1988; pp. 118–130.
58. Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549. [CrossRef]
59. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. Harmony search as a metaheuristic algorithm. *J. Heuristic* **2001**, *6*, 259–281. [CrossRef]
60. Mumford, C.L. Simple Population Replacement Strategies for a Steady-State Multi-objective Evolutionary Algorithm. In Proceedings of the Genetic and Evolutionary Computation—GECCO 2004, Seattle, WA, USA, 26–30 June 2004; Deb, K., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3102, pp. 1360–1371. [CrossRef]
61. Raidl, G.R.; Hu, B. Enhancing Genetic Algorithms by a Trie-Based Complete Solution Archive. In Proceedings of the Evolutionary Computation in Combinatorial Optimization (EvoCOP 2010), Istanbul, Turkey, 7–9 April 2010; Cowling, P., Merz, P., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6022, pp. 239–251. [CrossRef]
62. Wang, Q.; Sang, C.; Ma, H.; Wang, C. A nonrevisiting genetic algorithm based on multi-region guided search strategy. *Complex Intell. Syst.* **2025**, *11*, 22. [CrossRef]
63. Bui, T.N.; Moon, B.R. Genetic algorithm and graph partitioning. *IEEE Trans. Comput.* **1996**, *45*, 841–855. [CrossRef]

64. Salhi, S. Defining tabu list size and aspiration criterion within tabu search methods. *Comput. Oper. Res.* **2002**, *29*, 67–86. [CrossRef]
65. Talbi, E.G. A taxonomy of hybrid metaheuristics. *J. Heuristics* **2002**, *8*, 541–564. [CrossRef]